

4.3

*IBM i2 Analyze
Connector Developer Guide*



Note

Before you use this information and the product that it supports, read the information in [“Notices” on page 23](#).

This edition applies to version 4, release 3, modification 3 of IBM® i2® Analyze and to all subsequent releases and modifications until otherwise indicated in new editions. Ensure that you are reading the appropriate document for the version of the product that you are using. To find a specific version of this document, access the Configuring section of the [IBM Knowledge Center](#), and ensure that you select the correct version.

© **Copyright International Business Machines Corporation 2018, 2020.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. Connecting to external data sources.....	1
i2 Analyze and the i2 Connect gateway.....	1
System architecture with the i2 Connect gateway.....	2
System security with the i2 Connect gateway.....	4
Creating a connector to an external data source.....	4
Understanding the external source.....	5
Adding types to an i2 Analyze deployment.....	6
Writing a query with no parameters.....	7
Implementing a configuration endpoint.....	7
Adding a connector to the topology.....	10
Implementing an acquire endpoint.....	11
Modifying and testing a connector.....	12
Extending a system that queries external sources.....	13
Returning source references.....	13
Using types from different schemas.....	15
Converting records between types.....	16
Supporting seeded queries.....	19
Supporting parameters.....	20
Supporting validation.....	21
Notices.....	23
Trademarks.....	24

Chapter 1. Connecting to external data sources

Subject to your licensing terms, a deployment of i2 Analyze can use the i2 Connect gateway to query and retrieve data from external data sources. By implementing a connector to an external data source, you enable i2 Analyze to create and display records that represent the data in that source.

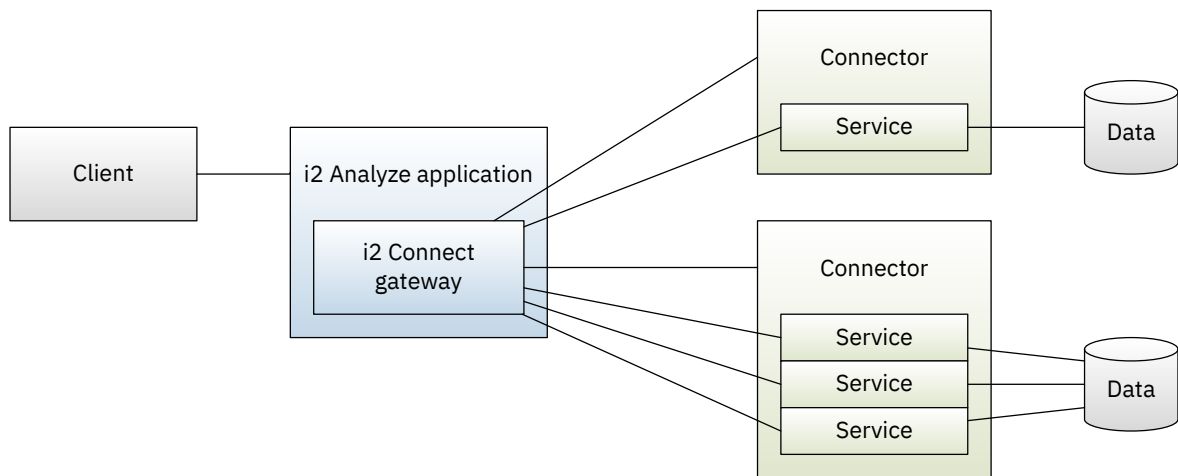
If the deployment and their permissions support it, users who create records from external sources can upload them to the Information Store for storage, sharing, and subsequent analysis.

Solutions that use the i2 Connect gateway depend on *connectors* that you create to make the bridge between i2 Analyze and external data sources. Conventionally, you write one connector for each source that you want to connect to. Each connector provides one or more *services* that present a querying interface to users and perform the actual queries on the data source.

i2 Analyze and the i2 Connect gateway

At version 4.3.3 of i2 Analyze, you can provide users with access to an Information Store, or enable them to query data in one or more external sources, or both. i2 Analyze deployments that are targeted at external data access include the i2 Connect gateway alongside or instead of the Information Store.

In a complete solution, the i2 Connect gateway provides i2 Analyze with information about what connectors and services are available and how clients are to use them. i2 Analyze passes client requests for data to the gateway, which receives and responds to the requests on behalf of those services.



Each connector contains the code that interacts with an external data source, the definitions of the services that it supports, and the descriptions of how clients might present its services to users. For example, clients need to know the following information:

- The name and description of each service
- Whether a service supports parameters that users can specify when they run it
- The input controls to display, and the validation to perform, for each parameter
- Whether a service behaves differently as a result of the current chart selection (Any selected records are then *seeds* for the operation that the service performs)

One of the roles of the gateway is to retrieve this information from all the connectors that i2 Analyze knows about. It can then help to ensure that requests to and responses from services are formatted correctly.

In solutions like these, a connector is an implementation of a REST interface that enables the gateway to perform its role. More specifically, a connector must support the following tasks:

- Respond to requests from the gateway for information about its services
- Validate and run queries against an external data source, including any queries that require parameters or seeds
- Convert query results to a shape that the gateway can process into i2 Analyze records

Writing a connector requires you to develop an implementation of the REST interface that the gateway expects, and to write queries that retrieve data from an external source. You must also understand the i2 Analyze data model, and be able to convert the retrieved data so that it conforms to an i2 Analyze schema.

The examples that IBM provides in the open-source repository at <https://github.com/IBM-i2/Analyze-Connect> demonstrate meeting several of these requirements in fully functional connectors. The remainder of this section explains how to meet them in more detail.

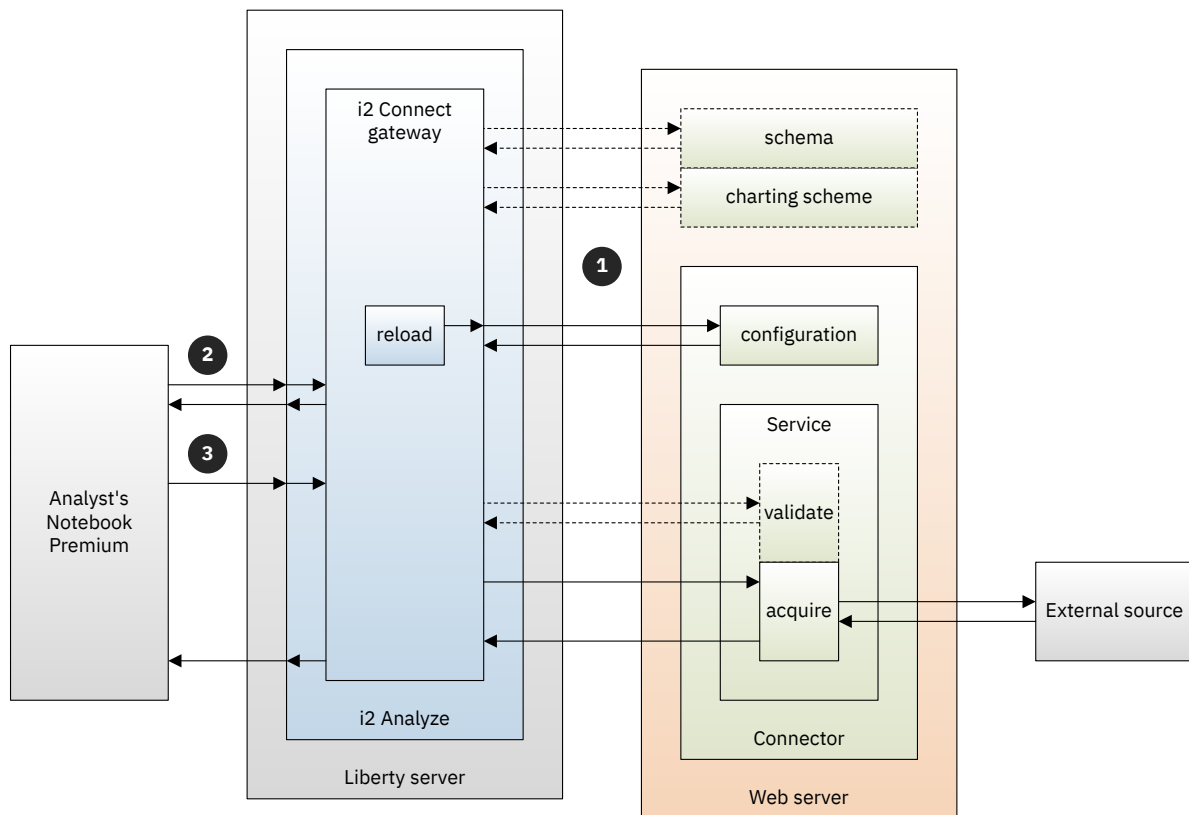
Note : For information on securing the communication between the i2 Connect gateway and its connectors, see [Client authenticated Secure Sockets Layer and the i2 Connect gateway](#).

System architecture with the i2 Connect gateway

The simplest possible deployment of i2 Analyze that uses connectors to access an external data source has four parts. The client, Analyst's Notebook Premium, communicates with a deployment of i2 Analyze that includes the i2 Connect gateway. The gateway manages at least one connector, which in turn exchanges data with an external source.

System interactions

The following diagram represents the deployment that is described in the introduction, and the interactions that take place between each part.



The diagram also identifies five of the REST endpoints that connectors can implement and use:

Configuration endpoint

The gateway sends a request to the mandatory configuration endpoint to gather information about all the services that the connector supports. All connectors have at least one service, and it is a service that implements the validate and acquire endpoints.

Schema endpoint

The schema endpoint is optional, and so is where you implement it. If a connector returns results whose types are not in the Information Store schema or a gateway schema, you can provide a connector schema that define those types. The configuration specifies whether and where a connector schema is available.

Charting scheme endpoint

The charting scheme endpoint is optional, and closely associated with the schema endpoint. A connector configuration that specifies a schema endpoint also specifies a charting scheme endpoint.

Validate endpoint

The validate endpoint is optional. If the configuration endpoint says that a particular service supports it, then the gateway sends a request to the validation endpoint immediately before a request to the acquire endpoint. The purpose of the validate endpoint is to test whether a request is reasonable before it is run.

Acquire endpoint

The acquire endpoint is mandatory for all services that provide synchronous querying. During a synchronous query, the gateway sends a request to the acquire endpoint that tells the service to query the external source. The service receives data back from the source and places it into the response that it returns to the gateway.

Note : For services that provide asynchronous queries, the `queriesResource` endpoint is mandatory instead of the acquire endpoint.

Interaction sequence

The interaction between the i2 Connect gateway and the other parts of an i2 Analyze deployment takes place in three distinct phases, labeled 1, 2, and 3 in the diagram.

1. When i2 Analyze starts up, the i2 Connect gateway sends a request to the configuration endpoint of every connector that is listed in the topology. It caches the information about the connectors and services that it receives in response.

If the configuration for any of the connectors specifies schema and charting scheme endpoints, the gateway sends requests to retrieve the connector schemas and charting schemes from those endpoints.

Note : You can also force the gateway to repeat this process while i2 Analyze is running. The gateway implements the [reload endpoint](#), whose POST method you can call after you change the configuration of a connector.

2. When a client connects to i2 Analyze, the latter gets the cached information from the gateway and returns it to the client. The client can then present the queries that the services implement to users, and help users to provide valid parameters and seeds to those queries where appropriate.
3. When a user runs a synchronous query, i2 Analyze passes it to the gateway for processing. The gateway packages the query into a request to the acquire endpoint on the service in question, preceded by a request to the validate endpoint if the service supports it.

On receiving the response from the acquire endpoint, the gateway converts the data that it contains into results that i2 Analyze can ultimately return to the client.

System security with the i2 Connect gateway

You can configure two different aspects of connector security in a deployment of i2 Analyze with the i2 Connect gateway. You can restrict which i2 Analyze users can run the queries that connectors implement, and you can secure the connection between the gateway and a connector.

Restrict access to a connector

You can use the command access control feature to control which users can use a connector. Through user groups, you can prevent users from using the i2 Connect gateway altogether, or you can restrict them to using only a subset of the deployed connectors. For more information, see [Controlling access to features](#).

Secure the connection to a connector

You can use client-authenticated SSL communication to secure the connection between the i2 Connect gateway and a connector. For more information, see [Client authenticated Secure Sockets Layer with IBM i2 Connect](#).

Creating a connector to an external data source

To create a connector, you need a data source to query, and a server that supports REST endpoints to query it from. On that foundation, you can build the functionality that takes requests from the i2 Connect gateway, retrieves data from the source, and returns it to the gateway in a shape that is ready for conversion to i2 Analyze records.

About this task

The easiest approach to writing a connector is to start by transferring data from the external source to the i2 Analyze server - and then to the client - with as few complications as possible. When you have that mechanism, you can use it as a base from which to develop the more complex services that you want to implement.

Note : The example connectors at <https://github.com/IBM-i2/Analyze-Connect> contain services that between them demonstrate several aspects of connector development. As you read this documentation, it is helpful also to read the source code for those examples.

Understanding the external source

Data is only valuable in i2 Analyze if users can examine it alongside, and with the same tools as, all the other data in the system. When you connect to an external source, you must decide how its contents relate to existing data and plan your connector development accordingly.

About this task

Before you write a connector, it is helpful to understand:

- What data is available from the external source
- How the available data maps to any schemas that you already have
- What your plans are for the finished connector

For example, if the purpose of your connector is to provide data from an external source to an existing deployment, it might be that some or all of the data in the source is similar to data that the deployment already models.

Alternatively, if you are writing a connector for use in many deployments, there are advantages to providing a description of the data that the connector returns alongside the connector itself.

During connector development, you can decide whether the definition for each type that you want to return is found in the Information Store schema, a gateway schema, or the connector schema. In the first two cases, you can also decide whether to use an item type that already exists or to extend the schema with a new type.

Procedure

1. Consider the type of the source and how you can query it.

For spreadsheets and text files, it is likely that you have complete access to the data they contain. For databases and web services, you might be restricted to a set of predefined queries.

2. Identify the types of data that you can retrieve from the source. If your aim is to integrate with an existing deployment, compare the source's types with the types in the Information Store or gateway schemas.

For each type in the source, you can reach one of three conclusions:

- The type and its data are a match for (or a subset of) one of the entity or link types you already have
- The type is a match for one of the types you already have, but its data implies new property types
- The type is not currently represented in the existing schemas

3. If the Information Store schema or a gateway schema already describes all the data in the external source, then you can start the process of writing queries that retrieve data from that source.

Note : In a deployment that contains more than one gateway schema, any single connector can use the types from only one gateway schema.

4. If the data in the external source contains *property* types that the existing schemas do not describe, then you can ignore that data or edit a schema to add those property types.

Additive changes to schemas are always permitted.

5. If the data in the external source contains *item* types that the existing schemas do not describe, then you can use one of the techniques for adding them to a schema, or create a connector schema to describe them.

Adding types to an i2 Analyze deployment

The data that you retrieve from an external source through the i2 Connect gateway must have entity and link types that are defined somewhere in the i2 Analyze deployment. If none of the existing schemas contains types that correspond with the external data, then you must extend or write a new schema as part of connector development.

Before you begin

If the data that you can retrieve from the external source is compatible with existing schema types, and your connector does not need to define its own types, then you can skip this task. Otherwise, you need to add type definitions to the i2 Analyze deployment.

About this task

There are two ways to add types to an existing deployment of i2 Analyze:

- Add the new entity, link, and property types directly to one of the deployed schemas in the usual way. Additive changes to schemas are always allowed.
- Create a gateway or connector schema (along with an accompanying charting scheme), and define the new item types in that schema.

Choosing which approach to use depends on your aims:

- Adding types to the Information Store schema makes them immediately and permanently available throughout the i2 Analyze deployment. Users can upload records that they retrieve from the external source to the Information Store without any further intervention.
- Especially in a deployment that does not include the Information Store, adding types to an existing gateway schema has some of the same benefits. The new types appear at familiar locations in the Analyst's Notebook Premium user interface, and they become available to future connectors.
- A key reason for creating a gateway schema to contain your new types is when you are creating a library of connectors that provide access to a particular source. The types in the gateway schema can be shared by all the connectors, and you can provide the gateway schema alongside the connectors if you need to distribute them.
- Creating a connector schema provides the highest level of flexibility. You can add the connector with its schema to any i2 Analyze deployment that includes the i2 Connect gateway. And you can make modifications to the connector schema without affecting the other schemas. But by its nature, this approach is also the least well integrated with the rest of the deployment.

Procedure

- If you decide to extend one of the deployed schemas, you can do so in the same way that you would add types for any other reason:
 - a) Open the XML file that contains the deployed schema in Schema Designer.
 - b) Add entity and link types (or property types) to the schema that correspond to data from the external source.
 - c) Add configuration for new and modified entity and link types to the charting scheme.
 - d) Update the deployment with the modified schema and charting scheme.For more information, see the [Schema Designer user guide](#), and the documentation on [modifying the Information Store schema](#) and [modifying gateway schemas](#).
- If you decide to create a schema, then you still use Schema Designer, but the process is different:
 - a) In Schema Designer, select **File > New Schema**.
 - b) Edit the schema and its charting scheme to add types for the data from the external source, and then save and close the XML files.

- c) To deploy a gateway schema, make it (and its charting scheme) available to the deployment toolkit, and then follow the instructions in [Adding, removing, and modifying gateway schemas](#).
- d) To deploy a connector schema, arrange for it (and its charting scheme) to be available from endpoints on the server that hosts the connector.
To complete this approach, you must tell the i2 Connect gateway where to find the endpoints that you created. You provide that information by [adding to your implementation of the configuration endpoint](#).

Writing a query with no parameters

All connectors contain at least one *service* that retrieves information from an external source. By starting with a service that runs a simple query, you can focus on aspects such as the REST endpoint implementations and data structures that are common to all services.

About this task

As you create a connector to an external source, writing a query that does not require parameters makes early testing and development easier. Not only does it simplify the connector, but also it simplifies running the query from the client.

Procedure

No fixed set of steps for retrieving data from external data sources exists, but some of your choices can make later tasks easier. Whatever the source, your initial aim is to generate a small set of representative data.

- If the source allows it, begin by writing a tightly bound query. Or, if your source comprises one or more text files, try to abbreviate them.
At the start of the process, large volumes of data can be distracting.
- If you can, make the query retrieve data for entities *and* links.
Creating the data structures that represent linked entities is an important part of connector development.
- Try to generate a data set that has most of the types that the external source contains.
No matter how complex your queries eventually become, the code to process data into the right shape for i2 Analyze is unlikely to change.

What to do next

When you [implement an acquire endpoint](#), you can call the query that you developed here directly from that code. Alternatively, during development, you might decide to run the query now, and save the results in a file.

Implementing a configuration endpoint

The configuration endpoint of a connector must respond to requests from the i2 Connect gateway with a JSON structure that describes the services in the connector. The more sophisticated the connector, the more information you need to provide.

Before you begin

Navigate to the source code for any of the connector projects in the `connector` directory of the [Analyze-Connect repository](#). In a text editor, open one of the `config.json` files that implements the configuration endpoint and demonstrates an approach that you might take.

About this task

When the i2 Analyze server starts, the i2 Connect gateway sends a request to the configuration endpoint. The response that your connector sends back tells the gateway what services are available. i2 Analyze caches this information and provides it to clients when they connect.

The simplest possible service does not support seeds and does not require users to provide parameters. As a result, it does not require the client to display user input controls. In that case, the structure that you return from the configuration endpoint must contain only two objects:

- The `defaultValues` object is mandatory.

Within the `defaultValues` object, you must specify the timezone for the i2 Connect gateway to apply to any retrieved value that does not specify its own timezone. There are also some optional settings:

- You can specify a direction for any retrieved link that does not specify its own direction.
- You can specify the identifiers and locations of the entity and link types that the i2 Connect gateway applies to records if a service does not indicate the types of data that it can retrieve.
- You can provide a default setting for whether the data identifiers that the connector returns are persistent from one call to the next.

Note : If you do not indicate whether these identifiers are persistent here or in the `services` array, the gateway assumes that the identifiers are *not* persistent.

- The `services` array is also mandatory.

The `services` array must contain information for at least one synchronous or asynchronous service. The information must include a unique identifier and a name for the service. It must also specify whether the service requires a client UI, and the URL of the `acquire` or `queriesResource` endpoint.

Procedure

First, construct the `defaultValues` object:

1. Determine the timezone that is most likely to apply to any temporal data in the source that the services in this connector query.
2. Find the identifier of the timezone in the IANA Time Zone Database, and arrange for the response from the endpoint to include the identifier in its `defaultValues` object.

For example:

```
{
  "defaultValues": {
    "timeZoneId": "Europe/London",
    ...
  },
  "services": [
    ...
  ]
}
```

Note : You can also retrieve a list of supported timezones from the GET method on the i2 Analyze server's `/api/v1/core/temporal/timezones` REST endpoint.

3. If the source contains only a few types, and if you intend the connector eventually to have many services that run different queries, then consider adding an `entityTypeId` and a `linkTypeId` to the `defaultValues` object.

It is more common to specify what types of record a query might retrieve on a per-service than on a connector-wide basis. However, if you do not supply default types here, then every service must supply types individually.

The gateway searches for the types that you supply here in the connector schema, the gateway schema, and the Information Store schema. To specify exactly where the types are defined, you can add `entityTypeLocation` and `linkTypeLocation` properties.

4. If you know that the source will always attach the same identifier to the same piece of retrieved information, add the `resultIdsPersistent` field to the `defaultValues` object and set its value to `true`.

Then, add an object to the `services` array:

5. To the `services` array, add a service object that has an `id` and a `name`. It is common to include a `description` and populate the `resultItemTypeIds` array as well, although neither is mandatory.

For example:

```
{
  "defaultValues": {
    "timeZoneId": "Europe/London",
    "resultIdsPersistent": true
  },
  "services": [
    {
      "id": "nypd-service",
      "name": "NYPD Connector: Get All",
      "description": "A service that retrieves all data",
      "resultItemTypeIds": {
        "INFOSTORE": ["ET1", "ET2", "ET3", "LT1", "LT2", "LT3"]
      },
      ...
    }
  ]
}
```

6. For a service whose query does not allow callers to provide parameters, you can set the `clientConfigType` to `NONE`.

```
{
  "defaultValues": {
    "timeZoneId": "Europe/London",
    "resultIdsPersistent": true
  },
  "services": [
    {
      "id": "nypd-service",
      "name": "NYPD Connector: Get All",
      "description": "A service that retrieves all data",
      "resultItemTypeIds": {
        "INFOSTORE": ["ET1", "ET2", "ET3", "LT1", "LT2", "LT3"]
      },
      "clientConfigType": "NONE",
      ...
    }
  ]
}
```

If you later [add parameters to the query](#), you can allow users to specify them by changing the value to `FORM` and [providing the identifier of a client configuration](#) in `clientConfigId`.

If you later [add support for seeds](#), you must add a `seedConstraints` object to the service.

7. Finally, for a synchronous service, set the `acquireUrl` of the service to the URL where you intend to host the `acquire` endpoint.

```
{
  "defaultValues": {
    "timeZoneId": "Europe/London",
    "resultIdsPersistent": true
  },
  "services": [
    {
      "id": "nypd-service",
      "name": "NYPD Connector: Get All",
      "description": "A service that retrieves all data",
      "resultItemTypeIds": {
        "INFOSTORE": ["ET1", "ET2", "ET3", "LT1", "LT2", "LT3"]
      },
      "clientConfigType": "NONE",
      "acquireUrl": "/all"
    }
  ]
}
```

Results

This JSON structure represents (almost) the simplest response that you can send from the configuration endpoint of a connector. It contains the definition of one simple service. In order for the i2 Connect gateway to retrieve that definition, you must add details of the connector to the i2 Analyze deployment topology.

Adding a connector to the topology

When you have an implementation of a configuration endpoint, you can tell i2 Analyze about the connector that it describes by adding it to the topology file. In that file, the `<connector>` element controls the behavior of the connector and the experience of its users.

Before you begin

Ensure that your deployment of i2 Analyze includes the i2 Connect gateway. If the gateway was not a part of the original deployment, you can add it by following the instructions in [Adding more data sources](#).

About this task

Each `<connector>` element in the topology file ties together the URL of a server, which might host several connectors, and the URL of the configuration endpoint for a particular connector. After you change the contents of the topology file, you must update the deployment and restart the i2 Analyze server.

Procedure

When you deploy i2 Analyze with one of the example configurations that includes the i2 Connect gateway, the topology file contains a `<connectors>` element. Adding a connector to the topology means adding a child to that element. (If your topology file does not contain a `<connectors>` element, you can also add it by hand.)

1. Navigate to the `configuration\environment` directory of the toolkit for your deployment of i2 Analyze. In a text editor, open the `topology.xml` file.
2. In the `<war>` element, locate the `<connector-ids>` child element, or add it if necessary.
3. Add a `<connector-id>` child element with the following format:

```
<connector-ids>
  <connector-id value="ConnectorId"/>
</connector-ids>
```

Here, *ConnectorId* must be unique within the topology file. Its purpose is to reference the configuration for the connector in the `<connectors>` element.

4. Add a `<connector>` element to the `<connectors>` element, which is a child of the root `<topology>` element:

```
<connectors>
  <connector id="ConnectorId"
    name="ConnectorName"
    base-url="Protocol://HostName:PortNumber"
    configuration-url="Protocol://HostName:PortNumber/Path"/>
</connectors>
```

Here, *ConnectorName* is likely to be displayed to users in the list of queries that they can run against external sources.

The i2 Connect gateway uses the value that you assign to the `base-url` attribute as the stem for the URLs that you specify in the configuration (such as `/exampleSearch/acquire`). `configuration-url` is the only optional attribute.

By default, the gateway attempts to retrieve the configuration from `<base-url>/config`. You can change this behavior by specifying a different URL as the value of `configuration-url`.

5. Save the file, and then update the deployed connectors configuration and restart the i2 Analyze server.

- a) On the command line, navigate to `toolkit\scripts`.
- b) Run the following commands in sequence:

```
setup -t stopLiberty
setup -t updateConnectorsConfiguration
setup -t startLiberty
```

When the server restarts, the i2 Connect gateway makes its request to the configuration endpoint. If that endpoint (or any of the endpoints in the retrieved configuration) is not available, the result is not an unrecoverable error. Rather, the i2 Analyze server logs the problem, and users see messages about unavailable services in the client application.

To preview what users see, and to discover the status of the configured connectors at any time, you can use the server admin console.

6. Open a web browser and connect to the i2 Analyze server admin console at `http://host_name/opal/admin`.

The admin console displays a list of connectors with their statuses. To see what a user with access to all connectors sees in their client application, click **Preview services**.

Implementing an acquire endpoint

In a connector, an acquire endpoint must perform two tasks in succession. Its roles are to retrieve data from an external source, and to return that data to the i2 Connect gateway in a form that the latter can use to create i2 Analyze records.

Before you begin

Refer again to the example connector projects at <https://github.com/IBM-i2/Analyze-Connect>. They contain implementations of the acquire endpoint for several services, which cover many of the common considerations.

About this task

When a user runs a synchronous query against an external data source through their client application, the i2 Connect gateway makes a request to the acquire endpoint of the relevant service. The request always has a *payload* that can contain information (often, parameter values or seed records) that the user provided to the query.

In general, your implementation of an acquire endpoint uses the contents of the payload to refine the commands that it sends to the external source. If the service that owns the endpoint specified a `clientConfigType` of `FORM`, then the identifiers of conditions in the form match the identifiers of conditions in the payload. If the service specified `seedConstraints`, then the payload contains the property values of seed records.

For a query with no parameters and no seeds, the payload is empty. You can focus your development effort on making sure that the response from the acquire endpoint meets the requirements of the i2 Connect gateway.

Procedure

Note : It is not possible to follow the same procedure for every different kind of external data source. The following steps do not apply in all cases. Rather, they describe tasks that you are likely to require.

1. In IBM i2 Analyze Schema Designer, open the schema file that contains the definitions of the i2 Analyze entity, link, and property types that data from the external source must align with.
2. Make a note of the identifiers and logical types of the property types of each of the entity and link types in the retrieved data. Make sure that you know which property types are mandatory.
3. Run the command to retrieve data from the external data source:
 - a) If data retrieval from the external source succeeds, store the data in a structure that you can manipulate easily, and continue with the next step in this procedure.

- b) If data retrieval from the external source fails, add the optional `errorMessage` field to the response from the endpoint.

The `errorMessage` field contains a string that users of the connector see when a problem occurs. When the field is present (and not empty), the operation fails and any data in the arrays of entities and links is ignored.

4. If the data from the external source contains information about links between entities, but the links in the data have no natural identifiers, you must manufacture those identifiers.
This problem is similar to the one you face if you [prepare data for ingestion](#) into the Information Store.
5. Create the arrays of entity and link data objects that must appear in the response from the acquire endpoint.
6. Populate the arrays according to the descriptions in the SPI documentation, taking care to ensure that you provide values for all mandatory properties.

Modifying and testing a connector

A connector to an external data source is defined by the information that the i2 Connect gateway retrieves from its configuration endpoint. To add a connector, or when you modify an existing one, you must arrange for the gateway to reload the information for all the connectors in the topology.

About this task

A connector that has only a configuration endpoint and an acquire endpoint is simple, but entirely functional. When you have a connector in this state, you can verify that many of the most important mechanisms are working correctly. To enable a faster test and development cycle, add new connectors to a running instance of i2 Analyze as soon as you can.

The i2 Connect gateway maintains a cache of configuration information that it provides to clients on request. Depending on the stage of development of your connector, you can use different approaches to make the gateway update its cache:

- If you add or edit a `<connector>` element in the topology file, then you must redeploy and restart the i2 Analyze server, as described in [“Adding a connector to the topology”](#) on page 10.
- If you modify a connector so that the response from its configuration endpoint changes - when you [add a validate endpoint](#), for example - you do not need to redeploy the server. Instead, you can either run the `restartLiberty` task from the deployment toolkit, or use the i2 Connect gateway's reload endpoint.

Regardless of how you update the gateway's cache of connector configuration information, the effect for clients and their users is the same. When a client next asks for a list of the services that it can use, any changes that you made are reflected in the list.

Procedure

To run `restartLiberty`, you can use the same technique as for any other deployment toolkit task. This section describes how to use the reload endpoint, and then explains some ways to diagnose problems with a new or updated connector.

The reload endpoint supports the POST method, which you can call through a command-line tool such as **postman** or **curl**. The endpoint requires authentication, which means that you must log in to the server and retrieve a cookie before you can POST to reload.

1. The following steps describe how to use **curl** to reload the connectors:
 - a) If the **curl** utility is not available on your server, download it from the project website at <https://curl.haxx.se/download.html>.
 - b) Open a command window and use **curl** to log in to the i2 Analyze server and retrieve an authorization cookie:

```
curl -i --cookie-jar cookie.txt
-d j_username=user_name
```

```
-d j_password=password  
http://host_name:host_port/context_root/j_security_check
```

This command connects to the specified i2 Analyze server as the specified user, retrieves the authentication cookie, and saves it to a local file named `cookie.txt`.

c) Use **curl** a second time to call the POST method on the reload endpoint:

```
curl -i --cookie cookie.txt  
-X POST http://host_name:host_port/context_root/api/v1/connectors/reload
```

This command causes the i2 Connect gateway to reload configuration information from all the connectors that the topology included when it was last deployed. Clients receive updated information about the connectors when they next connect to i2 Analyze.

2. After you reload the connectors (or restart the server), click **Preview services** in the server admin console to check that clients will reflect your changes in their lists of queries.

If you do not see your changes, different symptoms imply different causes:

- If a query does not appear in the list at all, then the problem lies with either the implementation of the configuration endpoint, or the `<connector>` element in the `topology.xml` file.
 - If a query is in the list but marked as unavailable, then the cause of the problem is either the implementation of the acquire endpoint, or the specification of that endpoint in the response from the configuration endpoint.
 - If the displayed information about a query is faulty or incomplete, look again at the definition of the corresponding service in the response from the configuration endpoint.
3. Provided that the queries due to new or modified connectors appear correctly, you can open a client, run them, and view the results that they return.

What to do next

After you test that your connector meets your requirements, you might choose to secure the connector in your deployment. You can control user access to the connector, and secure the connection between the connector and gateway. For more information, see [“System security with the i2 Connect gateway”](#) on page 4.

Extending a system that queries external sources

The simplest possible connector requires only a configuration endpoint and a basic acquire endpoint in order for clients to present its services to their users. By extending these endpoints and adding new ones, you can improve the reliability and utility of the connectors that you write.

Returning source references

Source references can provide users with rich information about where the data they see in i2 Analyze records came from. By adding source references to the record data that a service returns, you enable extra functionality in the client, including the ability to associate and view images with your records.

Before you begin

Source references provide up to five pieces of information about the source of an i2 Analyze record:

- The *type* of the source that data in the record came from
- The *name* of the source
- A *description* of the source
- The *location* of the source, which might be a URL
- An *image* of the source, which might also be an image for the record

When you return source reference information from a service, only the name is mandatory. However, you must also state whether users can edit or delete the source references that you add to your records.

Procedure

To add source references to the record data that your service returns to clients, you must arrange to include a `sourceReference` object as a peer of the `properties` object in your response from the `acquire` endpoint.

1. In your code that generates responses from the `acquire` endpoint, add at least the following extra content:

```
{
  ...
  "properties": {
    ...
  },
  "sourceReference": {
    "source": {
      "name": "source_name"
    }
  },
  ...
}
```

2. To give users the best experience, add a type and a description for your source that align with the definitions in the source reference schema for the deployment:

```
...
"sourceReference": {
  "source": {
    "name": "source_name",
    "type": "source_type",
    "description": "source_description"
  }
},
...
```

3. If the data in the external source has associated images, you can arrange for users to see an image for a record when they view it by adding an `image` field to the source object:

```
"source": {
  ...
  "image": "image_url"
}
```

You can similarly include the location of the source in the source reference by adding a `location` field and setting it to either a URL or a text description.

4. By default, when users add records from an external data source to a chart, they cannot edit or delete the source references that you add. To change that, add a `userModifiable` field to the `sourceReference` object:

```
...
"sourceReference": {
  "source": {
    ...
  },
  "userModifiable": true
},
...
```

Results

Completing these steps in your service code means that source references are present in the records that users retrieve and view. Source references are optional, so you can control whether to include them for all records, or just for records of particular types. Finally, source references have the same features in entity and link records, so you do not need to write different code for those two cases.

Using types from different schemas

Simply creating and deploying gateway and connector schemas does not automatically make the item types that they define available to your connectors. When you want to use gateway or connector schemas, you must specify them in the connector configuration, or the i2 Analyze deployment topology, or both.

Before you begin

These instructions assume that you already created gateway and connector schemas that define entity and link types for the data that your connector retrieves, as described in [“Adding types to an i2 Analyze deployment”](#) on page 6.

About this task

Anywhere a connector can specify an item type, it can also specify which schema that type is defined in: the Information Store, a gateway, or the connector schema. For the mechanism to work, the connector must provide information in its configuration about the gateway or connector schemas that it wants to use.

Procedure

Although a deployment of i2 Analyze can support several gateway schemas, an individual connector can use types from only one of them. You choose which gateway schema to use by adding its name to the response from the configuration endpoint.

1. At the top level of the configuration response, add and populate the `gatewaySchema` property.

```
{
  "gatewaySchema": "Example",
  "defaultValues": {
    ...
  },
  "services": [
    ...
  ]
}
```

Provided that a gateway schema with the short name EXAMPLE is configured in the `ApolloServerSettingsMandatory.properties` file for the deployment, the types in that schema are available for use in the connector.

To enable a connector schema, you must add endpoints for retrieving it to any connector that uses it, and then specify the locations of those endpoints in the response from the configuration endpoint.

2. Make your connector schema and its charting scheme available through two simple HTTP GET methods that do not require parameters.

It is easiest, but not mandatory, to implement these endpoints on the server that hosts the connectors that use them.

3. Add `schemaUrl`, `chartingSchemesUrl`, and `schemaShortName` settings to the response from your implementation of the configuration endpoint:

```
{
  "schemaUrl": "/schema",
  "chartingSchemesUrl": "/chartingSchemes",
  "schemaShortName": "Another-Example"
  "defaultValues": {
    ...
  },
  "services": [
    ...
  ]
}
```

In this form, i2 Analyze appends the URLs that you provide to the `base-url` that you specified for the connector in the topology file. By definition, these endpoints are on the same server as the connector. To use endpoints that are implemented elsewhere, you must provide the full URLs here.

Making changes like these affects the connector but not the topology, so you do not need to redeploy i2 Analyze. Restarting the server or using the reload endpoint on the i2 Connect gateway is enough.

4. Follow the steps in [“Modifying and testing a connector”](#) on page 12 to restart the server or force the gateway to reload your connector configuration.

The new types become available for use.

With the above changes in place, and provided that there are no identifier clashes, the connector can use item types from the connector, gateway, or Information Store schema just by using their identifiers. i2 Analyze searches for types in all three schemas, in that order. It is also possible to specify which schema a particular item type comes from:

- When you are specifying the default return types for the connector as a whole, you can set the `entityTypeLocation` and `linkTypeLocation` properties.
- When you are specifying the possible return types from a service, you can provide locations in the `resultItemTypeIds` structure.
- In each record that you return from a service, you can set the `typeLocation` property.

When you deploy a connector, you can use the `<connector>` element in the topology file to override some of the schema settings in the connector. You can instruct the connector to use a gateway schema with a different short name, which can be useful if you were unable to deploy the schema with the name that the connector expected. You can also change the short name of the connector schema, which is useful if you have two connectors that advertise the same name.

5. To specify a gateway schema with a different short name, add the `gateway-schema` attribute to the `<connector>` element:

```
<connectors>
  <connector id="ConnectorId"
             name="ConnectorName"
             base-url="Protocol://HostName:PortNumber"
             gateway-schema="ShortName"/>
</connectors>
```

6. To provide a different short name for the connector schema, add the `schema-short-name` attribute to the `<connector>` element:

```
<connectors>
  <connector id="ConnectorId"
             name="ConnectorName"
             base-url="Protocol://HostName:PortNumber"
             schema-short-name="NewShortName"/>
</connectors>
```

After you make changes to the topology of an i2 Analyze deployment, you must update the deployed connectors configuration and restart the i2 Analyze server, as described in [“Adding a connector to the topology”](#) on page 10.

Converting records between types

A deployment of i2 Analyze that defines a large number of item types across several schemas can become confusing for administrators and users, and might even compromise the utility of the platform. To reduce the number of types in the system, and to enable richer analysis, i2 Analyze supports the dynamic conversion of records from one type to another.

About this task

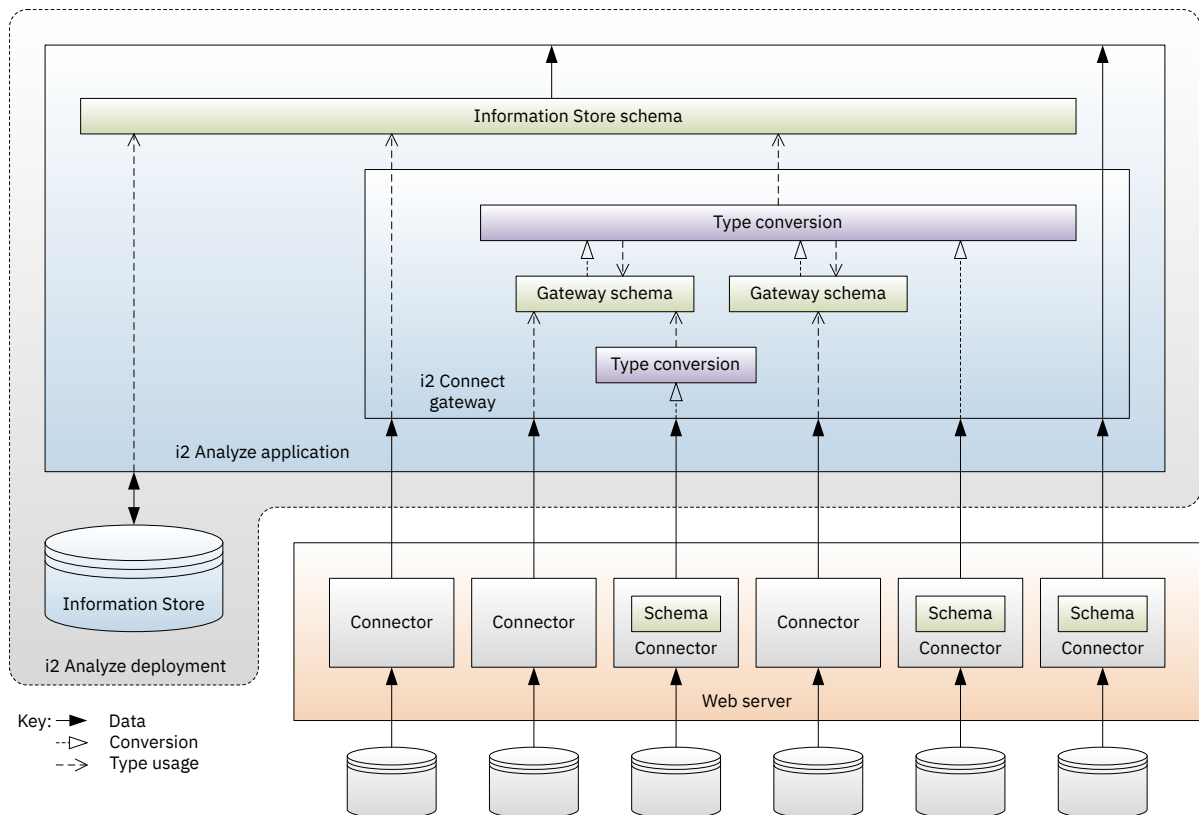
In a deployment of i2 Analyze that includes several schemas, and especially one that features connector and gateway schemas from third parties, there can be multiple item types with similar names or purposes. For example, two connector or gateway schemas might independently define similar "Person" item types, which might in turn be slightly different from a "Person" type defined in the Information Store schema.

This kind of arrangement can have a number of consequences:

- The existence of several item types with the same or similar names can be confusing for administrators and for users, who see records on their charts and item types in their palettes that look the same, but are not.
- By the same token, the analytical tools in i2 Analyze and IBM i2 Analyst's Notebook Premium do not consider records with similar-looking types to be equivalent for the purposes of comparison and other functions.
- Only records that have types from the Information Store schema can be uploaded to the Information Store.

To address these problems, the i2 Connect gateway provides a dynamic type conversion service. At startup, the gateway builds a catalog of all the item types in all the schemas in the deployment. You can then create mappings from one item type to another, in the following combinations:

- Connector schema type to Information Store schema type
- Connector schema type to gateway schema type
- Gateway schema type to Information Store schema type
- Gateway schema type to gateway schema type (in a different gateway schema)



Type conversion mappings describe how the property values of source records that have a particular item type become the property values of target records that have a different item type.

For example, a gateway schema might define an entity type named "Person", with property types including "Forename" and "Surname". In the same deployment, the Information Store might also define an entity type named "Person", with property types including "First Name" and "Family Name". Provided that the property types in question have compatible logical types, you can create a mapping to change the type of any record with the gateway type so that it becomes a record with the Information Store type.

The effect of this type conversion mapping is that any record that would otherwise appear in search results or on the chart surface with the gateway type instead appears with the Information Store type. As

a result, it can be compared directly with existing records in the Information Store, and can itself be uploaded to the Information Store if the user requires it.

Note : It is not mandatory to create mappings between item types. By default, records from connectors are copied to users' charts with their assigned entity or link types, without modification.

Procedure

To create type conversion mappings, you use the i2 Analyze Server Admin Console.

1. In a web browser, navigate to the URI of a i2 Analyze development environment, but append /admin. For example: `http://host_name:9082/opa1/admin`.

When you log in as a user with administrator privileges, the i2 Analyze Server Admin Console appears.

2. On the left of the user interface, select the **i2 Analyze Type Conversion** app.

In the main panel, the app displays all the item types from all the connector and gateway schemas in this deployment of i2 Analyze. (Item types from an Information Store schema cannot be source types in type conversion mappings.) For each type in the list, you can create a mapping to a different type.

3. Select a source item type, and click **Create mapping** to display the **Convert records** dialog.

This dialog displays all the item types from all the gateway schemas and any Information Store schema in this deployment of i2 Analyze. (Item types from connector schemas cannot be target types in type conversion mappings.)

4. Select a target type for your mapping, and click **Create mapping** to display the dialog with the same name.

The **Create mapping** dialog displays a list of the property types of the *target* item type. For each property type, you can elect to provide no mapping, or to say that a property with this type is to be set from a source property with a specified type, or that a property with this type is always to be set to a fixed value.

5. Configure the property type mappings as you see fit, and then click **OK** to create the item type mapping.

If there are problems with any of the mappings that you create, the app displays inline warning and error messages.

As you develop your type conversion mappings, you can preview their effects on the results that users see when they query external data sources by temporarily applying them to the development server.

6. Click **Apply** to apply all the current type conversion mappings to the server.
7. Click **Preview services** to display the same **Services** dialog that users see when they query external data sources.
8. Select and run some of the queries.

The descriptions and results of the queries now reflect the mapping that you created. Any references to the source types in your mappings are replaced with their target types.

This approach to previewing the effect of your mappings is not suitable for a production environment, where you must instead deploy a type conversion mapping configuration file to your i2 Analyze servers.

9. When you are satisfied with the mappings, click **Export** to create a configuration file named `mapping-configuration.json`.
10. Save the configuration file to the `configuration\fragments\common\WEB-INF\classes` folder of the i2 Analyze deployment toolkit.
11. Redeploy i2 Analyze to update the application with this change to its configuration. In a command prompt, navigate to the `toolkit\scripts` directory and run the following commands:

```
setup -t stopLiberty
setup -t deployLiberty
setup -t startLiberty
```

12. Finally, verify that the changes that you made in the development environment behave correctly in the production environment.

Supporting seeded queries

If you configure a connector's services to enable it, users can run queries with *seed records* that enable or change their functionality. Your implementations of the acquire endpoint receive identity and property data from selected records that you can use to drive operations against the external data source.

Before you begin

As you read the following information, also look at the `config.json` files in the example projects at <https://github.com/IBM-i2/Analyze-Connect>, some of which demonstrate the results of following this procedure.

About this task

The behavior of a query that supports parameters is modified by values that users specify when they run it. The behavior of a query that supports seeds is modified by the records that a user selects. Seeded queries generally fall into one of a handful of categories:

- A "find like this" query looks at the property values of a selected record and searches for data in the external source that has the same or similar property values.
- A "get latest" query uses the identifying information of a selected record to look for the same data in the external source, with the aim of updating the record.
- An "expand" query uses the identifying information of a selected record to search for data in the external source that is connected to that record.
- A "find path" query receives the identifying information for exactly two selected records and searches for a path that connects those records in the external source.

To indicate that a particular service supports seeds, you add a `seedConstraints` object to its definition in your response from the configuration endpoint. An empty object indicates that the service supports any number of *entity* seeds. To change this behavior, and for more control over the constraints, you can populate the object. The REST SPI documentation for the configuration endpoint describes its structure:

```
"seedConstraints": {
  "connectorIds": [""],
  "min": 0,
  "max": 0,
  "seedTypes": {
    "allowedTypes": "",
    "itemTypes": [
      {
        "id": "",
        "min": 0,
        "max": 0,
        "typeLocation": ""
      }
    ]
  }
}
```

A service can specify which records it accepts as seeds by restricting the connectors they came from. It can also set boundaries on the number of seed records it accepts, regardless of their type. A service can specify whether entity or link records are allowed as seeds (entity records are the default), and then restrict the range to a subset of that group. It can also set boundaries on the number of seeds, on a per-type basis.

If you configure the constraints so that requests must contain at least one seed of each permitted item type, then the service *requires* seeds. Users cannot run the service without an appropriate selection. Otherwise, your implementation of the acquire endpoint must support requests that might or might not contain seed information.

When the i2 Connect gateway calls your acquire endpoint to perform a seeded query, it includes a payload that contains identifiers and property values for all seeds. For information about the structure of the payload, see the REST SPI documentation for the `request` parameter of the acquire endpoint.

Procedure

In outline, the procedure for supporting seeded queries in one of your services is to start by adding seed constraints to its configuration. Clients then interpret the configuration and present the queries to users. When users run a query, your implementation of the acquire endpoint receives seeds that you can use to guide your response.

1. Decide what kind of seeded operation you want to perform, and its implications for the service.
For an "expand" query, for example, you might accept a fairly large number of entities of any type as seeds. For "find like this", the seed is more likely to be a single record of a specific type.
2. Add a `seedConstraints` object that reflects the requirements of the query, to the response from the configuration endpoint.
3. Implement the acquire endpoint for your service.

If at least one of the constraints does not specify a minimum record count, the endpoint might be called with or without seeds. The REST SPI documentation for the acquire endpoint describes the structure of the seed data.

Important : For "get latest" and "expand" queries, it is common for seed records also to appear in the results. In that case, you must ensure that the `id` of the outgoing record matches the `seedId` of the incoming record. In an "expand" query, for example, you can identify a seed as being one end of a link in the response by setting the `fromEndId` of the link to the `seedId`.

4. Restart the i2 Analyze server or instruct the i2 Connect gateway to reload the configuration. Then, connect to the server with a client and ensure that the service is working properly.

If your service supports being used with and without seeds, then its query appears in the client's lists of seeded and "unseeded" queries. Selecting the query in either list displays information about what seeds the user can or must select before they can use it.

Supporting parameters

In nearly all real situations, users want to be able to customize the queries that they can run against an external data source. To make that possible, you must configure the service to present parameters to its users. In your implementation of the acquire endpoint, you can act on the values that they specify.

Before you begin

As you read the following information, also look at the `config.json` files in the example projects at <https://github.com/IBM-i2/Analyze-Connect>, some of which demonstrate the results of following this procedure.

About this task

To support parameters in an service, you add a fixed set of conditions for which users can supply values. In a query for people, you might allow users to search for particular names or characteristics. In a "find path" seeded query, you might allow users to specify how long the path can be.

Note : A significant difference between visual query and external searches is that in the latter, conditions are not bound to property types. A condition in a service can be anything that you can process usefully in your implementation.

To add parameters to a service, you add a *client configuration* to its definition that describes how each condition is presented to users. The REST SPI for the configuration endpoint describes the structure of `clientConfig` objects, which look like this outline:

```
{
  "id": "",
  "config": {
    "sections": [
      {
        "title": "",
        "conditions": [
          {
```

```

        "id": "",
        "label": "",
        "description": "",
        "mandatory": false,
        "logicalType": ""
      }
    ]
  }
}

```

The client configuration is responsible for the appearance of the conditions that users see, and the restrictions on the values they can provide. As well as controlling the types of conditions, and specifying whether supplying a value is mandatory, you can add further validation that is appropriate to their type. For more information about this kind of validation, see the REST SPI documentation for the configuration endpoint.

When a user opens a query that supports parameters, they see a form that displays your conditions. When they provide values and run the query, your implementation of the acquire endpoint receives a payload that contains those values. You can write the implementation to act on those values in any way that makes sense for your data source.

Procedure

Like many other aspects of developing a connector, supporting parameters means changing your implementations of the configuration and acquire endpoints.

1. In your response from the configuration endpoint, write or modify the service definition so that its `clientConfigType` is "FORM", and add a `clientConfigId`.

This identifier links to a client configuration elsewhere in the response. In some circumstances, it might be appropriate to use the same client configuration for more than one service.

2. Add a `clientConfigs` array to the response, and within it a client configuration object whose `id` matches the identifier that you specified in Step 1.
3. Add your conditions to the client configuration.

To begin with, you might consider leaving out validation checks in the interests of getting a working implementation quickly.

4. Add code to your implementation of the acquire endpoint that unpacks the condition values from the payload.

You can then use those values to affect the query that you perform against the external source.

5. Restart the i2 Analyze server or instruct the i2 Connect gateway to reload the configuration.

You can now test the code and make sure that it does what you expect.

6. Return to the response from the configuration endpoint, edit the condition descriptions, and add client-side validation to improve the user experience.
7. Restart or reload again, and ensure that the validation has your intended effect.

What to do next

The validation that you can specify in the response from the configuration endpoint is performed by the client, and applies to values in isolation. If your service (and your users) might benefit from some more complex validation, consider adding a `validate` endpoint.

Supporting validation

The support in the i2 Connect gateway for queries that take parameters includes the ability to perform relatively simple, client-side validation on the values that users supply. In cases where running a query

might fail due to a set of values that are mutually incompatible, you can write an endpoint to perform server-side validation.

About this task

When you write a service definition that includes conditions, you have the option (and sometimes the duty) to include logic that validates the values that users supply for those conditions. For example, you might insist that a string is shorter than a certain length, or that its contents match a particular pattern. If you ask the user to select from a discrete set of values, then you must provide the values for them to select from.

However, there are other kinds of validation that the mechanism for defining conditions does not support. In particular, you cannot use it to validate values that are reasonable in isolation, but faulty in combination. (For example, dates of birth and death might both contain reasonable values, but it would make no sense to search for individuals where the latter is earlier than the former.)

If you have an implementation of the acquire endpoint for which it might be useful to perform this kind of validation, you can write a validate endpoint. When your configuration specifies a validate endpoint for a service, the gateway uses it before the acquire endpoint, and passes the same payload. If you decide that validation fails, the request to the acquire endpoint does not happen, and the user sees an error message that you specify.

Procedure

To add a validate endpoint to a service:

1. In your response from the configuration endpoint, add the `validateUrl` setting alongside the existing `acquireUrl` setting. Set its value to the location of the implementation.
2. Implement the rules for the validate endpoint in a way that is consistent with its definition in the REST SPI documentation.

The payload that the endpoint receives in the request contains all the same seed and parameter information as the acquire endpoint receives.

3. If validation succeeds according to your rules, return an empty response. If it fails, set the response to a simple object that contains an `errorMessage`:

```
{
  "errorMessage": ""
}
```

When the i2 Connect gateway receives a response that is not empty, it does not then send a request to the acquire endpoint.

4. Restart the i2 Analyze server or instruct the i2 Connect gateway to reload the configuration. Test that your new validation has the correct behavior.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Limited Hursley House Hursley Park Winchester, Hants, SO21 2JN UK

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.



Product Number: 5725-G22